Practical 1a

Aim:- Design an Expert system using AIML.

!pip install python-aiml

import aiml

kernel = aiml.Kernel()

kernel.learn("flu.aiml")

print("Expert System for Identifying Flu Symptoms")

print("Type 'bye' to exit the conversation.")

while True:

user\_input = input("You: ")

if user\_input.lower() == "bye":

print("System: Goodbye! Stay healthy.")

break

response = kernel.respond(user\_input.upper())

print(f"System: {response}")

Practical 1c

Aim:-Implement Conditional Probability and joint probability using Python.

import pandas as pd

df = pd.read\_csv('penguins.csv')

print("Data Preview:")

print(df.head())

pivot\_table = pd.crosstab(df['species'], df['island'], normalize=True)

print("\nJoint Probability (Pivot Table):")

print(pivot\_table)

conditional\_probability = pivot\_table.div(pivot\_table.sum(axis=0),

axis=1)

print("\nConditional Probability of Species given Island:")

print(conditional\_probability)

normalized=True

print("\nJoint Probability is represented in the pivot table (Species vs Island):")

print(pivot\_table)

p\_adelie\_given\_biscoe = conditional\_probability.loc['Adelie', 'Biscoe']

print(f"\nP(Adelie | Biscoe) = {p\_adelie\_given\_biscoe:.4f}")

Practical 2a (Prolog Code)

Aim:- Create a simple rule-based system in Prolog for diagnosing a common illness based on symptoms.

%Facts:Define symptoms

symptom(fever).

symptom(cough).

symptom(sore\_throat).

symptom(body\_aches).

symptom(runny\_nose).

symptom(headache).

symptom(fatigue).

%Facts:Define possible illnesses

condition(cold).

condition(flu).

condition(strep\_throat).

%Rules: Diagnosing based on the presence of symptoms

diagnose(cold):-

symptom(runny\_nose),

symptom(cough),

symptom(sore\_throat),

\+ symptom(fever). %Absence of fever

diagnose(flu):-

symptom(fever),

symptom(cough),

symptom(body\_aches),

symptom(headache),

symptom(fatigue).

diagnose(sterp\_throat):-

symptom(sore\_throat),

symptom(fever),

\+symptom(cough). %Absence of cough

%Alternative:Diagnosing based on rule covering all possible symptoms

diagnose(unknown):-

\+diagnose(cold),

\+diagnose(flu),

\+diagnose(strep\_throat).

%Assuming the patient has the following symptoms:

symptom(fever).

symptom(cough).

symptom(body\_aches).

symptom(headaches).

symptom(fatigue).

%You can ask Prolog:

?-diagnose(Condition).

Practical 2b

Aim:- Design a Fuzzy based application using Python.

import numpy as np

import skfuzzy as fuzz

from skfuzzy import control as ctrl

import matplotlib.pyplot as plt

traffic\_density = ctrl.Antecedent(np.arange(0, 101, 1), 'traffic\_density')

time\_of\_day = ctrl.Antecedent(np.arange(0, 25, 1), 'time\_of\_day')

green\_light\_duration = ctrl.Consequent(np.arange(0, 61, 1), 'green\_light\_duration')

traffic\_density['low'] = fuzz.trimf(traffic\_density.universe, [0, 0, 50])

traffic\_density['medium'] = fuzz.trimf(traffic\_density.universe, [30, 50, 70])

traffic\_density['high'] = fuzz.trimf(traffic\_density.universe, [50, 100, 100])

time\_of\_day['non\_peak'] = fuzz.trimf(time\_of\_day.universe, [0, 0, 12])

time\_of\_day['peak'] = fuzz.trimf(time\_of\_day.universe, [10, 24, 24])

green\_light\_duration['short'] = fuzz.trimf(green\_light\_duration.universe, [0, 0, 20])

green\_light\_duration['moderate'] = fuzz.trimf(green\_light\_duration.universe, [15, 30, 45])

green\_light\_duration['long'] = fuzz.trimf(green\_light\_duration.universe, [40, 60, 60])

traffic\_density.view()

time\_of\_day.view()

green\_light\_duration.view()

rule1 = ctrl.Rule(traffic\_density['low'] & time\_of\_day['non\_peak'], green\_light\_duration['short'])

rule2 = ctrl.Rule(traffic\_density['low'] & time\_of\_day['peak'], green\_light\_duration['moderate'])

rule3 = ctrl.Rule(traffic\_density['medium'] & time\_of\_day['non\_peak'], green\_light\_duration['moderate'])

rule4 = ctrl.Rule(traffic\_density['medium'] & time\_of\_day['peak'], green\_light\_duration['long'])

rule5 = ctrl.Rule(traffic\_density['high'] & time\_of\_day['non\_peak'], green\_light\_duration['long'])

rule6 = ctrl.Rule(traffic\_density['high'] & time\_of\_day['peak'], green\_light\_duration['long'])

green\_light\_ctrl = ctrl.ControlSystem([rule1, rule2, rule3, rule4, rule5, rule6])

green\_light\_sim = ctrl.ControlSystemSimulation(green\_light\_ctrl)

green\_light\_sim.input['traffic\_density'] = 75 # High traffic

green\_light\_sim.input['time\_of\_day'] = 18 # Peak hours

green\_light\_sim.compute()

print(f"Recommended Green Light Duration: {green\_light\_sim.output['green\_light\_duration']} seconds")

green\_light\_duration.view(sim=green\_light\_sim)

plt.show()

Practical 3a

Aim:- Simulate genetic algorithm with suitable example using Python any other platform.

import random

import string

target\_string = "HELLO"

population\_size = 50

mutation\_rate = 0.01

generations = 200

def fitness(individual):

return sum(1 for a, b in zip(individual, target\_string) if a == b)

def create\_population(size):

return [''.join(random.choices(string.ascii\_uppercase, k=len(target\_string))) for \_ in range(size)]

def select\_parents(population):

tournament = random.sample(population, 5)

return max(tournament, key=fitness)

def crossover(parent1, parent2):

crossover\_point = random.randint(1, len(parent1) - 1)

return parent1[:crossover\_point] + parent2[crossover\_point:]

def mutate(individual):

individual = list(individual)

for i in range(len(individual)):

if random.random() < mutation\_rate:

individual[i] = random.choice(string.ascii\_uppercase)

return ''.join(individual)

population = create\_population(population\_size)

for generation in range(generations):

best\_individual = max(population, key=fitness)

print(f"Generation {generation}: Best individual: {best\_individual}, Fitness: {fitness(best\_individual)}")

if fitness(best\_individual) == len(target\_string):

break

new\_population = []

for \_ in range(population\_size):

parent1 = select\_parents(population)

parent2 = select\_parents(population)

child = crossover(parent1, parent2)

child = mutate(child)

new\_population.append(child)

population = new\_population

best\_individual = max(population, key=fitness)

print(f"Best individual: {best\_individual}, Fitness: {fitness(best\_individual)}")

Practical 3b

Aim:- Design intelligent agent using any AI algorithm. design expert tutoring system

class MathTutor:

def \_\_init\_\_(self):

self.operations = {

'+': lambda a, b: a + b,

'-': lambda a, b: a - b,

'\*': lambda a, b: a \* b,

'': lambda a, b: a/b,

}

def explain\_operation(self, operator):

explanation = {

'+': "Addition adds two numbers together.",

'-': "Subtraction subtracts the second number from the first.",

'\*': "Multiplication gives the product of two numbers.",

'': "Division divides the first number by the second.",

}

return explanation.get(operator, "Invalid operation.")

def perform\_operation(self, operator, a, b):

if operator in self.operations:

return self.operations[operator](a, b)

else:

return None

if \_\_name\_\_ == "\_\_main\_\_":

tutor = MathTutor()

# Example usage:

operator = '+'

a, b = 10, 5

print(tutor.explain\_operation(operator))

result = tutor.perform\_operation(operator, a, b)

print(f"Result of {a} {operator} {b} = {result}")